Udriştoiu Stefan
Stoica Spahiu Cosmin

Lucrarea 9 – Structuri arborescente.

Introducere

Dacă ne-am uitat la un arbore genealogic, sau la o ierarhie de comandă într-o firmă, am observat informaţiile aranjate într-un *arbore.*

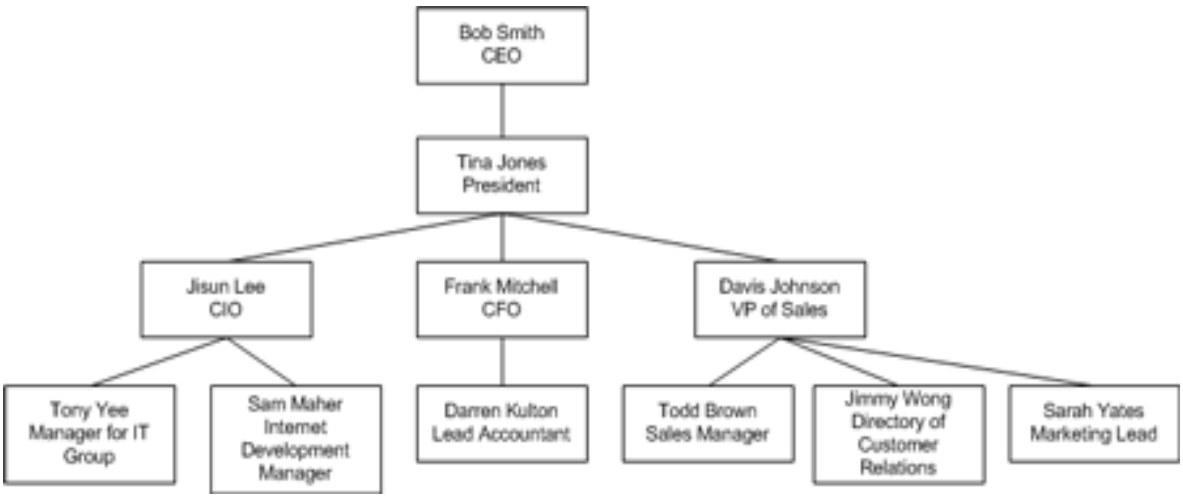
Un arbore este compus dintr-o colecţie de *noduri*, unde fiecare nod are asociată o anumită informaţie şi o colecţie de copii.

Copiii unui nod sunt acele noduri care urmează imediat sub nodul însăşi.

Părintele unui nod este acel nod care se află imediat deasupra.

Rădăcina unui arbore este acel nod unic, care nu are nici un părinte.

Exemplu de ierarhie de comandă intr-o firmă:



In acest exemplu rădăcina arborelui este Bob Smith. Este rădăcină deoarece nu are nici un părinte. Copilul său direct este Tina Jones.

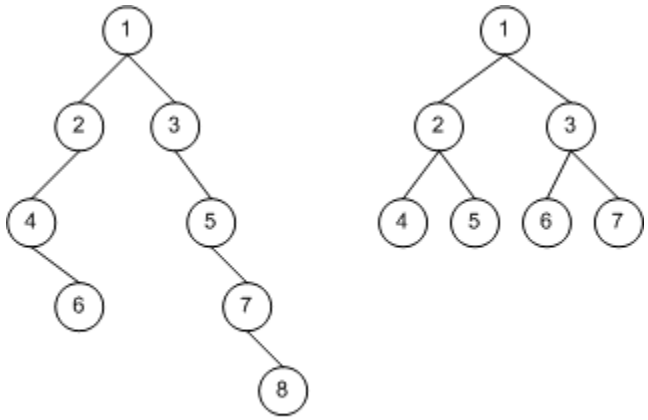
Tina Jones are 3 noduri copii (nodurile de pe nivelul următor din ierarhie): Jisun Lee, Frank Mitchell şi Davis Johnson. Părintele său este Bob Smith (nodul de pe nivelul superior, din ierarhie).

Toţi arborii au următoarele proprietăţi:

- Există o singură rădăcină
- Toate nodurile, cu excepţia rădăcinii, au exact un singur părinte
- Nu există cicluri. Aceasta înseamnă că pornind de la un anumit nod, nu există un anumit traseu pe care îl putem parcurge, astfel încât să ajungem înapoi la nodul de plecare.

Arbori Binari

Arborii binari sunt un tip aparte de arbori, în care fiecare nod are maxim 2 copii. Pentru un nod dat, într-un arbore binar, vom avea copilul din stânga, şi copilul din dreapta. Exemplu de arbori binari:



Arborele din figura a) are 8 noduri, nodul 1 fiind rădăcina. Acesta are ca şi copil stânga nodul nr.2, iar ca şi copil dreapta nodul nr.3. La rândul său nodul nr.2 nu are decât un singur copil (stânga), şi anume nodul nr 4.

Deci un nod dintr-un arbore binar poate avea 2 copii (stânga şi dreapta), un singur copil (doar stânga sau doar dreapta) sau nici unul (exemplu nodul 8).

Nodurile care nu au nici un copil se numesc noduri frunză.

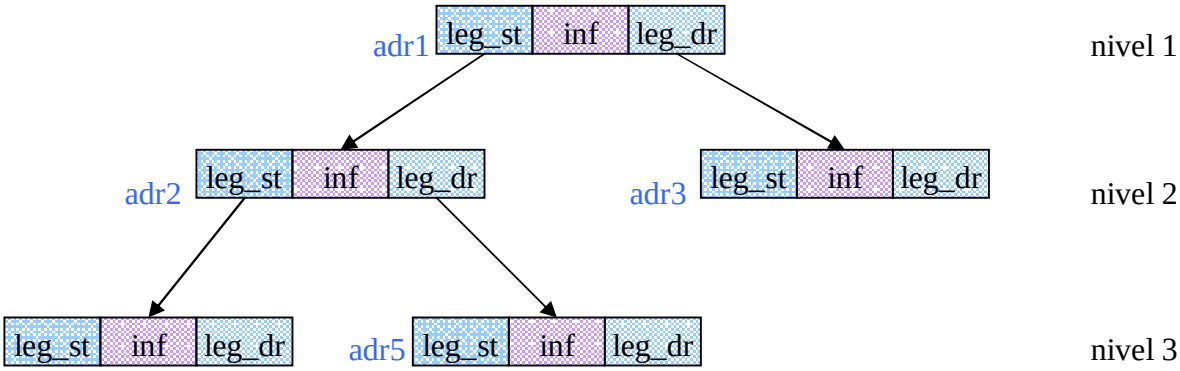
Nodurile care au 1 sau 2 copii se numesc noduri interne.

Pentru reținerea informației în calculator se va folosi alocarea dinamică. Deoarece pentru fiecare nod în parte este necesar să se rețină, pe lângă informația utilă şi legăturile cu nodurile copii (adresele acestora), ne putem imagina următoarea structura a unui nod:


```
struct Nod {
      int inf;
      Nod *leg_st;
      Nod *leg_dr;
};
Nod *prim;
```

unde: *leg_st* reprezintă adresa nodului copil din stânga, *inf* reprezintă câmpul cu informație utilă, iar *leg_dr* este legătura cu copilul din dreapta.

Un arbore binar va avea următoarea structură internă:



Pe nivelul 1 vom avea un singur nod, nodul rădăcină. Componenta *leg_st* va fi egală cu adresa adr2, nodul din stânga de pe nivelul 2, iar componenta *leg_dr* va fi egală cu adresa adr3.

Componentele *leg_st* şi *leg_dr* ale nodului din stânga de pe nivelul 2, vor fi egale cu adr4, respectiv adr5.

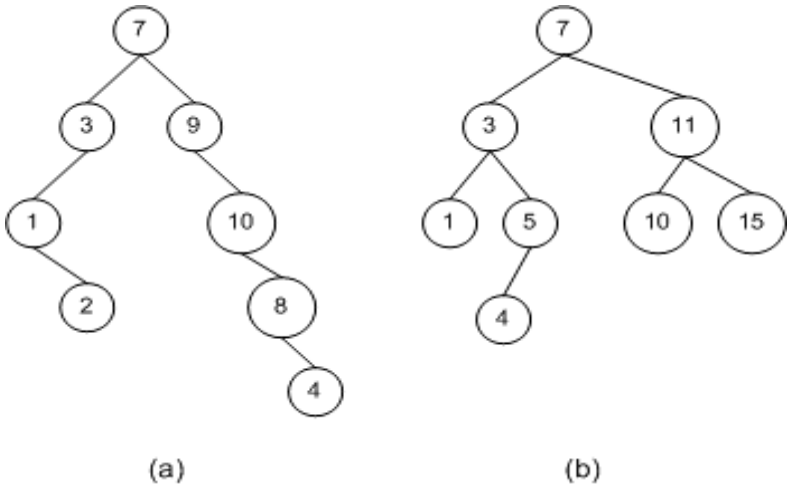
Nodul din dreapta de pe nivelul 2 nu mai are nici un copil şi de aceea componentele sale *leg_st* şi *leg_dr* vor avea valoarea NULL (ca şi la liste dinamice).

În mod analog şi componentele *leg_*st şi *leg_dr* ale nodurilor de pe nivelul 3 vor avea valoarea NULL.

Arbori Binari de Căutare

Un arbore binar de căutare este un arbore binar destinat îmbunătăţirii timpului de cutarea informaţiei. El va trebui să respecte următoarea proprietate: pentru oricare nod *n*, fiecare din descendentii din subarborele din stânga va avea valoarea informaţiei mai mică decât a nodului *n*, iar fiecare din descendenţii din subarborele din dreapta va avea valoarea informaţiei mai mare decât a nodului n.

Exemplu:



în figura de mai sus avem 2 arbori binari.

Arborele din figura b) este arbore binar de căutare deoarece este respectată regula de mai sus, si anume că în oricare subarbore stâng valorile trebuie să fie mai mici decât ale rădăcinii şi în oricare subarbore dreapta, valorile trebuie să fie mai mari decât ale rădăcinii. (Cu alte cuvinte, toate valorile din nodurile aflate în stânga nodului n trebuie să fie mai mici, iar cele din dreapta mai mari).

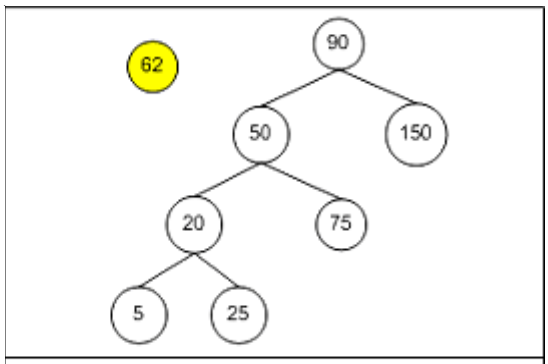
Se foloseşte exprimarea „oricare din subarborii" stâng respectiv drept, deoarece pentru rădăcina 7, elementele subarborelui stâng sunt 3, 1, 5, 4 (care sunt toate mai mici decât 7), iar elementele subarborelui drept sunt 11, 10, 15 (care sunt toate mai mari decât 7). Dacă în schimb vom considera ca rădăcină nodul 3, vom avea ca subarbore stâng nodul 1 (1 este mai mic decât 3), iar ca subarbore drept nodurile 5 şi 4 (ambele mai mari decât 3).

Dacă vom considera ca şi rădăcină pe nodul 10, acesta nu are nici un copil, deci implicit respectă regula pentru arborii binari de căutare.

Arborele din figura a) nu este un arbore binar de căutare deoarece nu toate nodurile respectă regula.

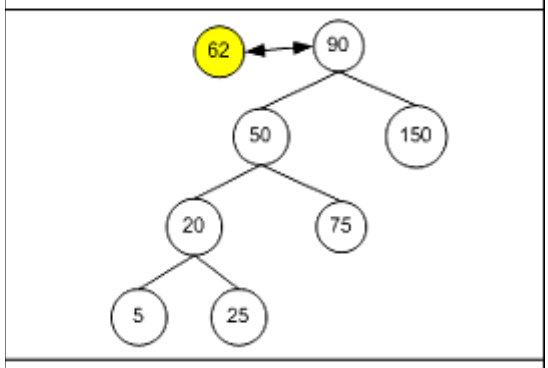
Nodul 4 nu are ce căuta acolo în dreapta nodului 8. Ar trebui să se afle în stânga lui 8 (4<8). Dacă privim mai sus observăm că nodul 4 ar trebui să se afle şi in stânga nodului 10, şi în stânga nodului 9, şi în stânga nodului 7. Cu alte cuvinte nodul 4 nu are ce căuta în subarborele drept. Unde ar trebui să fie poziţionat astfel încât şi arborele din figura a) să fie arbore binar de căutare?

Crearea şi inserarea într-un arbore binar de căutare



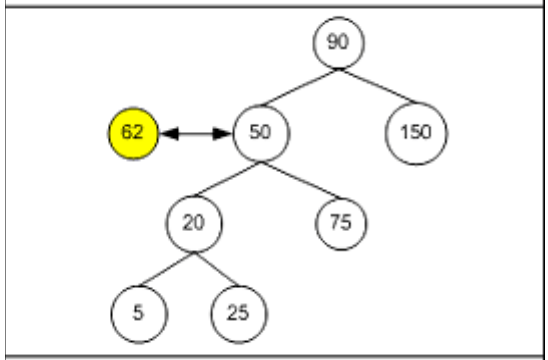
Vom considera arborele din dreapta. Să presupunem că dorim să inserăm nodul cu valoarea 62. Acesta se va insera ca nod frunză.

Pentru a-l insera va trebui să căutăm o poziție în arbore care respectă regula de integritatea a arborilor binari de căutare.

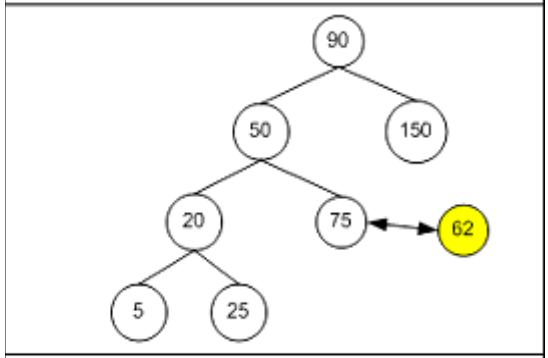


Vom începe prin compararea nodului de inserat (62), cu rădăcina arborelui (90).

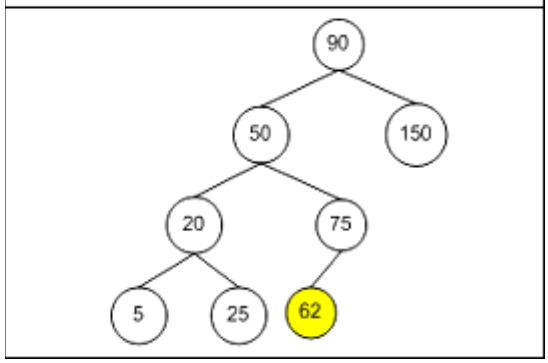
Observăm că este mai mic decât ea, deci va trebui inserat undeva în subarborele stâng al acesteia.



Vom compara apoi 62 cu 50. Din moment ce 62 este mai mare decât 50, nodul 62 va trebui plasat undeva în subarborele drept al lui 50.



Se compară apoi 62 cu 75. Deoarece 75 este mai mare decât 62, 62 trebuie să se afle în subarborele din stânga al nodului 75



Dar 75 nu are nici un copil în partea stângă. Asta înseamnă că am găsit locația pentru nodul 62. Tot ceea ce mai trebuie făcut este să modificăm în nodul 75 adresa către copilul din stânga, încât să indice spre 62.

Prin crearea unui arbore binar de căutare (A.B.C.) vom înțelege de fapt crearea rădăcinii, restul nodurilor fiind adăugate prin operația de inserare.

Implementare :

```
 //programul va primi ca parametru numărul pe care trebuie să îl adauge
             void insertie(int nr)
             {
                 Nod  *nod1, *nod2, *nod3;


//se crează nodul care se adaugă în arbore.
                 nod1 = new Nod;
                 nod1->inf = nr;
//deoarece o să se adauge ca nod frunză, el nu va avea nici un copil, deci legăturile stânga şi dreapta, vor fi nule
                 nod1->leg_st = NULL;
                 nod1->leg_dr = NULL;
//verificam mai intai dacă există o rădăcină (dacă arborele a fost creat)
              if(prim==NULL)
              {
//nodul creat va deveni rădăcina arborelui
               prim = nod1;
              }
//dacă există un nod rădăcină, inserarea în arbore se va face conform algoritmului prezentat mai sus.
              else
              {
                 nod2 = prim;
//se va căuta locul de inserare al nodului nod1, pornind căutarea din rădăcină. Nodul nod3 va reprezenta nodul părinte al nodului nod1
                 while (nod2 != NULL) {
                       if (nr < nod2->inf) {
                             nod3 = nod2;
                             nod2 = nod2->leg_st;
                       }//end if
                       else { // se merge spre dreapta
                             nod3 = nod2;
                             nod2 = nod2->leg_dr;
                       }//end else
                 }//end while
//după găsirea nodului părinte, se crează legătura nod3 (nod părinte) -> nod1 (nodul inserat acum)
                 if (nr < nod2->inf)
//se asaza in stânga părintelui
                       nod3->leg_st = nod1;
                 else
//se aşază în dreapta părintelui
                       nod3->leg_dr = nod1;
               }//end else
               return t;
             }
```

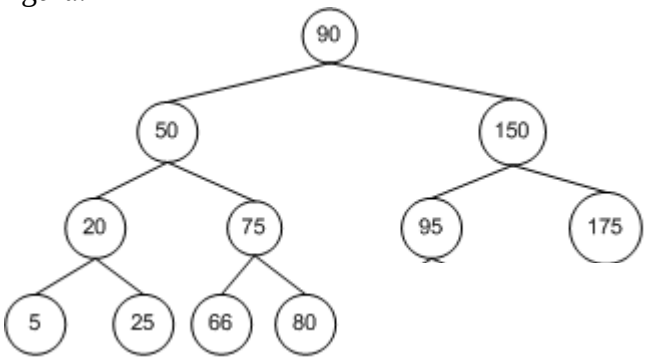
Parcurgerea unui Arbore (Listarea)

Există 3 tipuri de parcurgere a unui arbore: inordine, preordine şi post ordine.
Aceste denumiri corespund modului cum este vizitată rădăcina.
Parcurgerea în preordine:

Parcurgerea în inordine se bazează pe următorul principiu: se va vizita mai întâi rădăcina, copilul (copiii) din stânga, iar apoi copilul (copiii) din dreapta.

Implementarea se va face recursiv, astfel încât nu ne interesează ce se întâmplă decât pe unul din nivele (de exemplu pe primul), pe restul procedându-se identic.

Fie arborele din figură:



Parcurgerea în preordine presupune:

1. se va vizita **rădăcina 90**
2. se vizitează copilul din stânga (50). Dar acesta este la rândul său rădăcină pentru subarborele 20 – 75. Fiind rădăcină, ea se vizitează prima. Deci vizităm şi **nodul 50.**
3. se vizitează copilul din stânga al nodului rădăcină 50, adică nodul 20. Si acesta este rădăcină pentru subarborele 5 – 25. Fiind rădăcină, se va vizita şi **nodul 20**.
4. se vizitează copilul din stânga al rădăcinii 20, adică **nodul 5**. Si acesta este rădăcină pentru arborele vid. Il vizităm.
5. cum pe ramura aceasta nu mai avem ce vizita, ne întoarcem la rădăcina 20. Aceasta este deja vizitată. La fel şi copilul din stânga. Vom vizita copilul din dreapta, **nodul 25.**
6. nodul 25 nu mai are copii, deci nu mai avem ce vizita sub el. Cum pentru nodul rădăcină 20 am vizitat si copilul din stânga şi copilul din dreapta, vom mai urca un nivel la nodul rădăcină 50. Pentru acesta tocmai am terminat de vizitat subarborele stâng. Mai avem de vizitat subarborele drept. Vom merge la **nodul 75**. Nodul 75 este rădăcină pentru un alt subarbore. Îl vizităm.
7. mergem la copilul din stânga, **nodul 66**.
8. cum nodul 66 nu mai are copii, revenim la rădăcina 75, pentru care vom vizita copilul din dreapta: **nodul 80**
9. pentru rădăcina 75 am vizitat tot ce se putea. Urcăm la rădăcina 50. Si pentru aceasta am vizitat tot ce se putea. Urcăm la rădăcina 90. Aici tocmai am terminat de vizitat subarborele stâng. Mai avem de vizitat subarborele drept. Vom merge la **nodul 150**. El este rădăcină pentru subarborele 95 – 175, deci îl vizităm
10. vizităm copilul său din stânga, **95**
11. Ne întoarcem la rădăcina 150. Vizităm copilul din dreapta, **175**
12. ne întoarcem la rădăcina 90, pentru care am terminat de vizitat si subarborele drept. Cum nu mai există nici un nivel deasupra sa, am terminat de vizitat tot arborelel.

Vizitarea arborelui va întoarce vectorul:

90 – 50 – 20 – 5 – 25 – 75 – 66 – 80 – 150 – 95 – 175

Deşi la prima vedere pare destul de complicat, implementarea este destul de simplă. Se va folosi o funcţie Preordine care primeşte ca parametru iniţial rădăcina arborelui (respectiv, prin apelări succesive, rădăcinile subarborilor)

```
void Preordine(Nod* rad)
{
```

```
//dacă nu s-a ajuns la ultimul nod
        if (rad != NULL) {
//se vizitează rădăcina
            printf(" %d  - ",rad->inf);
// se vizitează copilul din stânga, apoi cel din dreapta
            Preordine(rad->leg_st);
            Preordine(rad->leg_dr);
          }
       }
```

Parcurgerea în inordine

Parcurgerea în inordine presupune vizitarea mai întâi a copilului din stânga, apoi vizitarea rădăcinii şi mai apoi a copilului din dreapta.

Pentru arborele anterior, se procedează astfel:

1. se porneşte de la rădăcina 90. Pentru aceasta se vizitează mai întâi copilul din stânga, nodul 50. Si acesta este la rândul lui o rădăcină pentru arborele 20 - 75. De aceea vom vizita mai intai copilul său stâng, nodul 20, care la rândul său este rădăcină pentru arborele 5 – 25. Vom vizita mai întâi copilul stâng, **nodul 5.**
2. pentru nodul 5 nu mai avem ce vizita. Revenim la **rădăcina 20**, pentru care tocmai am vizitat subarborele stâng. Conform definiției, o vizităm pe ea.
3. vizităm **nodul 25**, copilul său drept.
4. revenim la rădăcina 20. Nu mai avem ce vizita pe acest nivel. Urcăm la **rădăcina 50**. Pentru aceasta am vizitat subarborele stâng. Este rândul ei să o vizităm.
5. Vom vizita apoi subarborele ei drept, după modelul anterior

……..

În final vom obține parcuregerea:

5 – 20 – 25 – 50 – 66 – 75 – 80 – 90 – 92 – 95 – 111 – 150 – 166 – 175 – 200

Se observă faptul că parcurgerea în inordine va afişa de fapt vectorul ordonat crescător.

```
        void Inordine(Nod* rad)
        {
//dacă nu s-a ajuns la ultimul nod
         if (rad != NULL) {
// se vizitează copilul din stânga
            Inordine(rad->leg_st);
//se vizitează rădăcina
            printf(" %d  - ",rad->inf);
//se vizitează copilul din dreapta
            Inordine(rad->leg_dr);
         }
        }
```

Parcurgerea în postordine

Pentru parcurgerea în postordine, se va vizita mai întâi subarborele stâng, apoi subarborele drept şi de abia ultima dată rădăcina.

Care este vectorul returnat de această parcurgere?

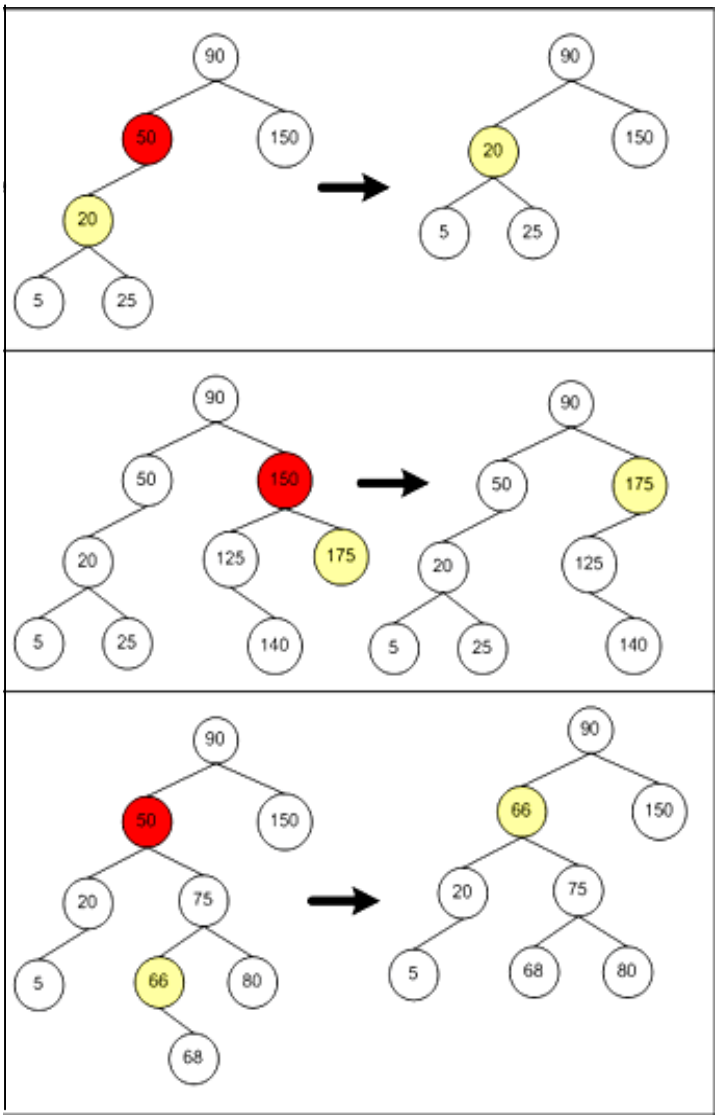
Implementați algoritmul!

Ştergerea dintr-un arbore binar de căutare

Ideea de bază pentru ştergerea dintr-un arbore astfel încât să se păstreze structura de arbore binar de căutare este să înlocuim nodul care se doreşte să se şteargă cu *cel mai din*

stânga nod, al subarborelui drept al nodului de şters.

Pentru ştergerea unui nod avem următoarele cazuri:



Cazul I

Dorim să ştergem nodul 50. Acesta nu are subarbore drept, aşa că îl vom înlocui pur şî simplu cu nodul 20.

Cazul II

Dorim să ştergem nodul 150.

Subarborele drept nu conţine decât nodul 175 Nu există un subarbore stâng al subarborelui drept.

Se va înlocui nodul 150 cu 175.

Cazul III

Dorim să ştergem nodul 50.

Deoarece subarborele drept al nodului 50 conţine un subarbore stâng, se va alege cel mai din stânga nod al subarborelui drept a lui 50. Acest cel mai din stânga nod va conţine *cel mai mic număr mai mare decât nodul de şters*. In cazul nostru acest nod este 66.

Prezentăm în continuare implementarea algoritmului de mai sus. Funcţia de ştergere va primi ca parametru informaţia care dorim să o ştergem (numărul de şters) şi va returna true în cazul în care a făcut ştergerea sau fals în cazul în care informaţia care dorim să o ştergem din arbore, nu a fost găsită (dorim să ştergem nodul 1000?).

```
bool Sterge(int nr)
{
    Nod *tmp,*tmp1,*tmp2;
//se porneşte căutarea informaţiei de şters din rădăcină
    tmp1 = rad;
    while (tmp1->inf != nr)
    {
        if (tmp1->inf > nr)
            tmp1 = tmp1->leg_st;
        else
            tmp1 = tmp1->leg_dr;
    }
    tmp = tmp1;
//dacă suntem în cazul I
```

```
        if (tmp1->leg_dr==NULL){
//mutare inf din nodul din st, in nodul curent
                tmp1->inf=tmp1->leg_st->inf;
//refacere legături
                tmp1->leg_st = tmp1->leg_st->leg_st;
                delete tmp1->leg_st;
        }
        else
//nu avem copil stânga
           if (tmp1->leg_st == NULL){
//mutare inf din nodul din st, in nodul curent
                        tmp1->inf=tmp1->dr->inf;
                        tmp1->dr = tmp1->dr->dr;
                        delete tmp1->dr;
                }
//cazul III
                else {
//mergem la copilul din dreapta
                        tmp = tmp1->leg_dr;
                        tmp2 = tmp1;
//căutăm cel mai din stânga copil
                        while (tmp->leg_st != NULL)
                        {
                                tmp2 = tmp;
                                tmp = tmp->leg_st;
                        }

                        tmp1->inf = tmp->inf;
                        tmp2->leg_st = NULL;
                        delete tmp;
                }//end else

   return 0;
   }
```

Probleme propuse

1. Implementați operațiile de adăugare, ştergere din arbore, căutare şi parcurgere prin toate trei metodele, folosind doar funcții recursive. Implementați o funcție de parcurgere care sa returneze sirul ordonat descrescător. (a nu se ordona vectorul după ce s-a făcut parcurgerea)

2. Implementați operațiile de adăugare în arbore (recursiv), ştergere din arbore (nodul de șters se va înlocui cu cel mai din dreapta nod al subarborelui stâng al nodului de șters) și parcurgere Parcurgerea se va face printr-o metodă care să returneze sirul ordonat descrescător.

3. Implementați un arbore unde fiecare nod are maxim 3 copii, aranjați după următoarea regulă:
   - copilul din stânga este mai mic decât jumatate din nodul rădăcină (nod x < rădăcină/2)
   - nodul din mijloc are informația cuprinsă între [rădăcină/2 si rădăcină]
   - nodul din dreapta are informația > decât informația din rădăcină

4. Evidența produselor aflate în stocul unui magazin se ține într-un arbore de căutare. Pentru fiecare nod se reține: denumire produs, preț, cantitate. În fiecare seară arborele este actualizat prin comenzi de forma:
   i → se mai introduc produse primite de la depozit (Atenție: un astfel de produs este posibil să se mai afle în stoc)
   s → se șterg produsele vândute în ziua respectivă
   v → se tipărește valoarea tuturor produselor aflate pe stoc
   L → se tipărește o listă cu toate produsele aflate pe stoc

5. Se citesc de la tastatură n numere naturale care se adaugă pe măsură ce se citesc într-o stivă (implementată dinamic). Să se adauge aceste numere într-un arbore binar de căutare.

6. Definim un arbore de căutare pentru numere complexe, în care fiecare nod va conține ca informație utilă partea reală şi partea imaginară a unui număr complex de forma x + iy. Să se creeze un arbore binar de căutare care să memoreze n numere complexe citite de la tastatură, apoi să se scrie o funcție care să șteargă din arbore numerele complexe care au partea reală şi partea imaginară egală.

7. Un arbore binar de căutare se numeşte AVL dacă pentru orice nod, diferența dintre înălțimea subarborelui stâng şi a subarborelui drept este -1, 0 sau 1 (înălțimea reprezintă numărul de nivele). Să se construiască un arbore AVL care să aibă în noduri cheile 1, 2, … n. Indicație: să se folosească un algoritm Divide et Impera. La fiecare pas al procedurii recursive de creare se introduce în arbore un subinterval al mulțimii {1,2 … n} cuprins între {st…m} iar apoi {m+1, … dr}, cu m = (st + dr)/2

8.